CT484: PHÁT TRIỂN ỨNG DỤNG DI ĐỘNG

XÂY DỰNG ỨNG DỤNG MYSHOP - PHẦN 4

File báo cáo cần nộp là file PDF trong đó có ghi thông tin **mã sinh viên, họ tên, lớp học phần** cùng với **hình minh họa tại các bước kiểm tra kết quả thực thi, các chức năng (không cần chụp hình mã nguồn)**. Cuối file báo cáo ghi đường link đến GitHub mã nguồn của dự án.

Ứng dụng MyShop có các chức năng chính sau:

- Hiển thị và cập nhật danh mục các sản phẩm
- Xem chi tiết một sản phẩm
- Đánh dấu sản phẩm được yêu thích
- Thêm, xóa sản phẩm trong giỏ hàng
- Thực hiện đặt hàng, xem lại đơn hàng, chi tiết đơn hàng
- Đăng ký, đăng nhập
- Lưu trữ dữ liệu trên Firebase

Sinh viên chỉ cần tạo MỘT báo cáo duy nhất cho tất cả các buổi thực hành.

(Tiếp tục từ kết quả phần 3)

File báo cáo buổi 4 làm tiếp tục từ file báo cáo buổi 3.

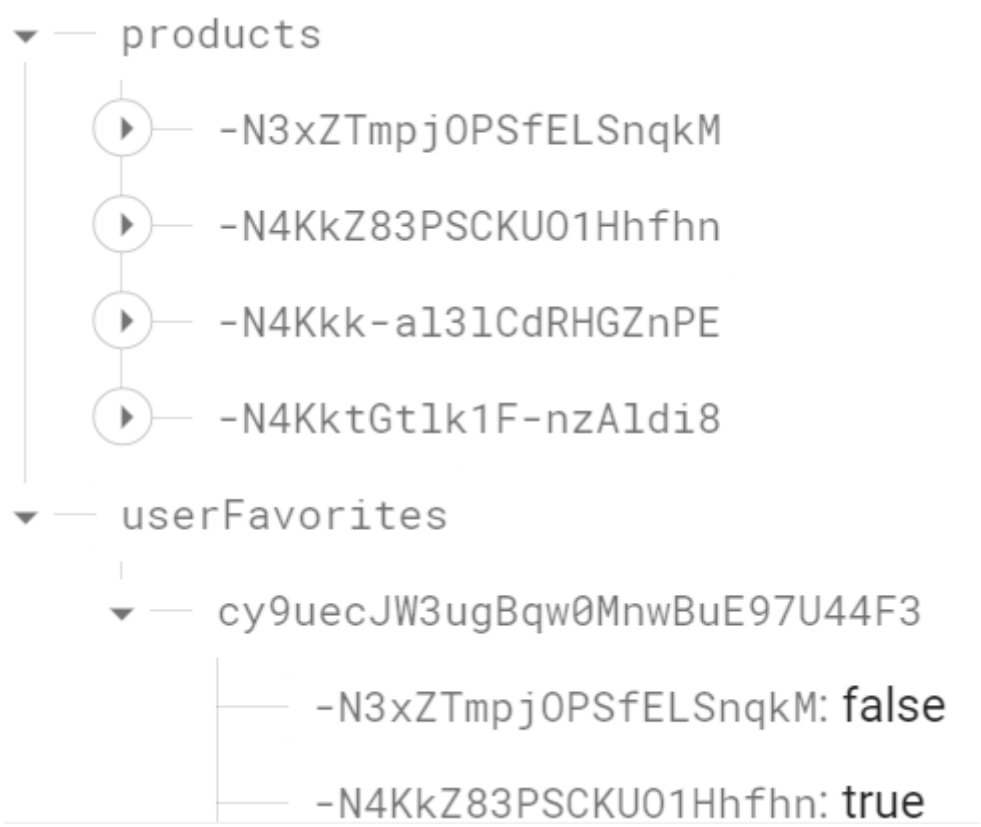
Bước 0: Tìm hiểu về Firebase

Xem video hướng dẫn về Firebase tại đây: https://www.youtube.com/watch?v=robC2XADUkQ

Firebase là một tập hợp các dịch vụ lưu trữ (hosting service) cho nhiều dạng ứng dụng khác nhau (Android, iOS, Node.js, PHP, ...). Firebase hỗ trợ dịch vụ lưu trữ cơ sở dữ liệu (CSDL) NoSQL (Firebase Database), lưu trữ tập tin (Cloud Storage), chức năng như một dịch vụ (Cloud Functions), chứng thực (Firebase Authentication), thông báo (Firebase Cloud Messaging), ... Dự án này chỉ sử dụng dịch vụ CSDL và dịch vụ chứng thực của Firebase. Các dịch vụ này đều có hạn mức dùng miễn phí.

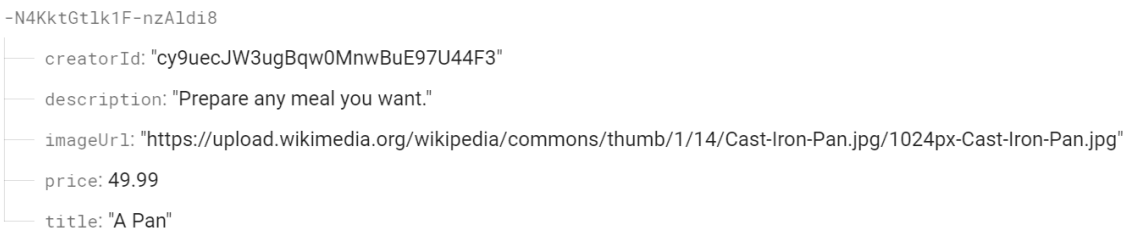
Firebase Database

Có hai tùy chọn cho CSDL trên Firebase: Realtime Database và Cloud Firestore. Realtime Database là dạng CSDL nguyên thủy của Firebase trong khi Cloud Firestore là phiên bản mới hơn. Nội dung bài thực hành này trình bày cách sử dụng Realtime Database. Mô hình lưu trữ Realtime Database khá đơn giản: CSDL được tổ chức như một **cây JSON**. Dữ liệu thêm vào cây JSON này sẽ trở thành một nhánh trong cây. Tên nhánh trong cây có thể được chỉ định tường minh hoặc tự động sinh:



Có thể sử dụng thư viện client Firebase dành cho Flutter hoặc dùng REST API để tương tác với Realtime Database. Để đơn giản và không làm mất đi tính tổng quát, REST API sẽ được sử dụng. Các URI tham chiếu đến các đối tượng trong cây JSON được tạo ra như sau: *đính kèm đuôi ".json" vào đường dẫn đến đối tượng cần truy xuất*, ví dụ:

- **GET /products.json**: đọc tất cả các đối tượng từ nhánh products
- **GET /products.json?orderBy="creatorId"&equalTo="cy9uecJW3ugBqw0MnwBuE97U44F3"**: đọc tất cả các đối tượng từ nhánh products có trường creatorId = cy9uecJW3ugBqw0MnwBuE97U44F3 (cần tạo chỉ mục cho trường creatorId, xem cấu hình rule bên dưới)
- **POST /products.json**: thêm một đối tượng mới vào nhánh products. Tên nhánh ứng với đối tượng mới sẽ được tự động sinh và trả về trong câu trả lời
- **GET /products/N4KktGtlk1F-nzAldi8.json**: đọc thông tin sản phẩm có định danh N4KktGtlk1F-nzAldi8 (nhánh /products/N4KktGtlk1F-nzAldi8). Nếu nhánh này có cấu trúc như sau:

  thì dữ liệu trả về sẽ ở dạng:

```
{
    "creatorId": "cy9uecJW3ugBqw0MnwBuE97U44F3",
    "description": "Prepare any meal you want.",
    "imageUrl": "https://upload.wikimedia.org/.../1024px-Cast-Iron-Pan.jpg",
    "price": 49.99,
    "title": "A Pan"
}
```

- **GET /userFavorites/cy9uecJW3ugBqw0MnwBuE97U44F3.json**: đọc danh mục các sản phẩm yêu thích của người dùng có định danh cy9uecJW3ugBqw0MnwBuE97U44F3. Theo như hình ví dụ phía trên, dữ liệu trả về sẽ có dạng như sau:

```
{
    "-N3xZTmpjOPSfELSnqkM": false,
    "-N4KkZ83PSCKUO1Hhfhn": true
}
```

Để đảm bảo an toàn/bảo mật, cần cấu hình các luật truy xuất trên Realtime Database (ví dụ chỉ cho phép những yêu cầu cập nhật từ những người dùng được chứng thực).

Firebase Authentication

Firebase Authentication cung cấp các dịch vụ backend, SDK dễ sử dụng (cũng có hỗ trợ REST API), các thư viện UI sẵn dùng để giúp chứng thực người dùng cho ứng dụng. Nhiều cơ chế chứng thực được hỗ trợ: chứng thực dựa trên mật khẩu, số điện thoại, mạng xã hội như Google, Facebook, Twitter, ...

Chúng ta sẽ sử dụng chứng thực với email và mật khẩu, cụ thể sẽ sử dụng hai lời gọi REST sau đây:

- Đăng ký người dùng mới:
  - Phương thức yêu cầu: POST
  - Endpoint: `https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=[API_KEY]`, với API_KEY là "Web API Key" lấy từ dự án firebase
  - Dữ liệu JSON gửi đi:

```
{
    "email":"[user@example.com]",
    "password":"[PASSWORD]",
    "returnSecureToken":true
}
```

  - Dữ liệu JSON trả lời (thành công):

```
{
    "idToken": "[ID_TOKEN]",
    "email": "[user@example.com]",
    "refreshToken": "[REFRESH_TOKEN]",
    "expiresIn": "3600",
    "localId": "tRcfmLH7..."
}
```

- Đăng nhập người dùng
  - Phương thức yêu cầu: POST
  - Endpoint: `https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=[API_KEY]`, với API_KEY là "Web API Key" lấy từ dự án firebase
  - Dữ liệu JSON gửi đi:

```
{
    "email":"[user@example.com]",
    "password":"[PASSWORD]",
    "returnSecureToken":true
}
```

- Dữ liệu JSON trả lời (thành công):

```
{
  "localId": "ZY1rJK0eYLg...",
  "email": "[user@example.com]",
  "displayName": "",
  "idToken": "[ID_TOKEN]",
  "registered": true,
  "refreshToken": "[REFRESH_TOKEN]",
  "expiresIn": "3600"
}
```

Khi người dùng hoặc thiết bị đăng nhập bằng Firebase Authentication, Firebase sẽ tạo một ID token định danh người dùng (**idToken**) và nhờ đó có thể truy cập vào một số tài nguyên, chẳng hạn như Realtime Database và Cloud Firestore, i.e., có thể sử dụng ID token đó để xác thực REST API của Realtime Database. (Tuy nhiên, gán quyền truy xuất dữ liệu cụ thể phải thông qua cấu hình các luật truy xuất).

Gửi ID token như chuỗi truy vấn trên URI đến REST API của Realtime Database, ví dụ: **GET /products.json?auth=<ID_TOKEN>**.

Để hỗ trợ các trường hợp chứng thực nâng cao hơn ví dụ như *điều khiển truy cập dựa trên vai trò người dùng*, có thể tham khảo giải pháp tạo token tùy biến hoặc gán thông tin tùy biến cho tài khoản người dùng. Các giải pháp này yêu cầu có thêm xử lý phía backend (tự tạo hoặc dùng Firebase Functions) để có thể cài đặt được. Một giải pháp không yêu cầu xử lý backend (dùng trực tiếp Firebase Console) là tạo nhánh cây lưu thông tin người dùng, trong đó có chỉ định vai trò của người dùng và định nghĩa các luật truy cập tài nguyên dựa trên thông tin vai trò đó, ví dụ:

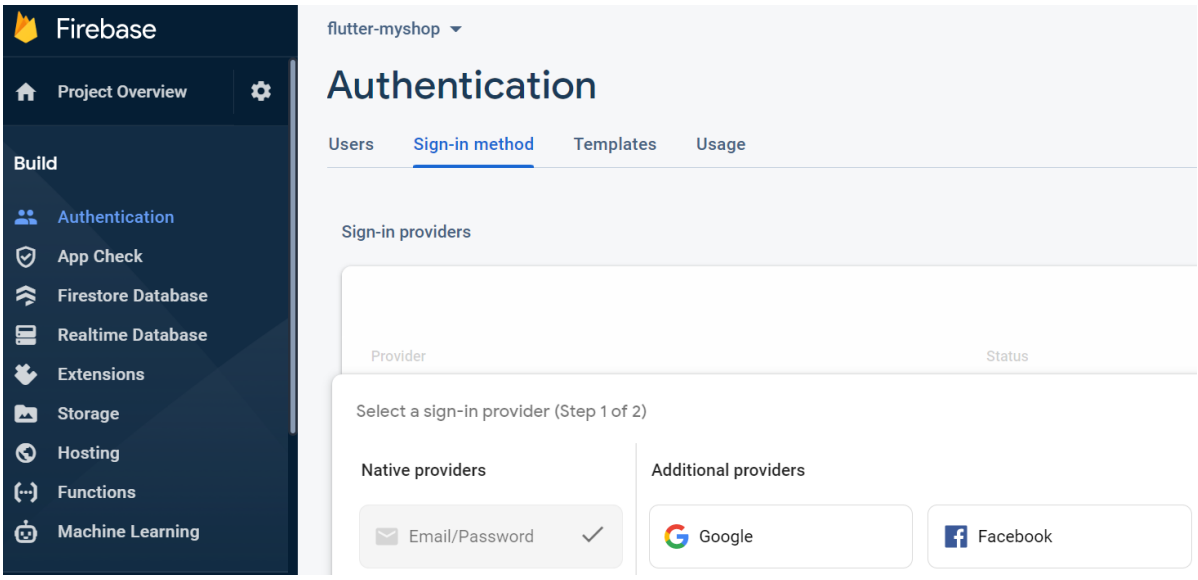
```
{
  "rules": {
    ".read": true,
    "users": {
      ".write": "root.child('users').child(auth.uid).child('role').val() ==
'admin'"
    }
  }
}
```

Nhánh cây users có dạng như sau:

```
users
|
--- cy9uecJW3ugBqwOMnwBuE97U44F3
    |
    ---role: "admin"
```

Bước 1: Thiết lập dự án trên Firebase

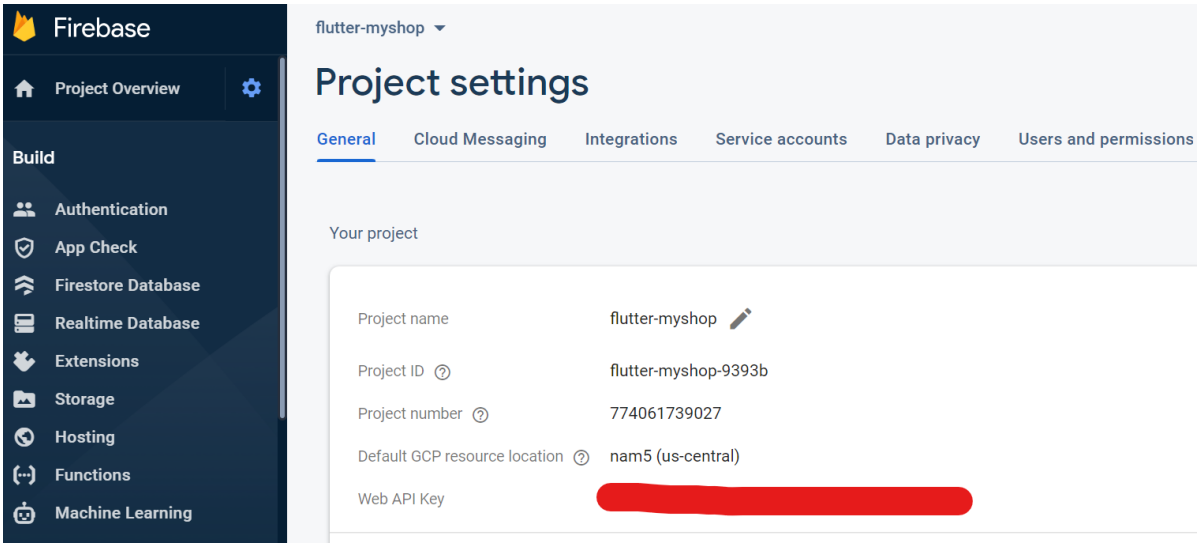
- Tạo dựa án trên Firebase (https://console.firebase.google.com/), ví dụ tên là *flutter-myshop* (trong quá trình tạo dự án, ngoài tên dự án có thể chỉ định ID của dự án hoặc để tự động sinh).
- Bật chế độ chứng thực người dùng dạng email/password trên Firebase Authentication:



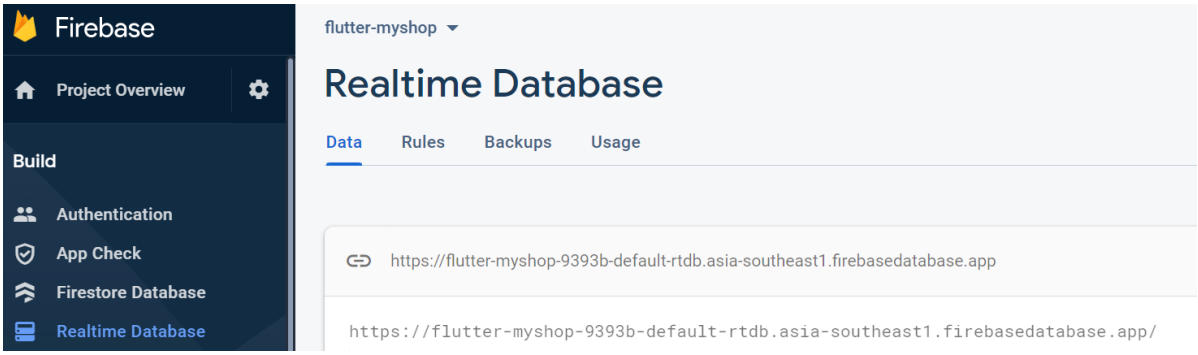
- Trong thư mục gốc của dự án, tạo tập tin *.env* và lưu hai biến sau:

```
FIREBASE_API_KEY=<Web API key từ dự án Firebase>
FIREBASE_RTDB_URL=<URL đến CSDL Realtime trên Firebase>
```

trong đó FIREBASE_API_KEY được lấy từ Project settings:



FIREBASE_RTDB_URL lấy từ Realtime Database:



- Chuyển qua thẻ "Rules" và thêm vào các luật truy cập dữ liệu như sau (chọn "Publish" để lưu):

Các luật trên có ý nghĩa như sau: tất cả các người dùng hợp lệ (auth.uid != null) đều có thể đọc dữ liệu. Trên nhánh /products (lưu danh mục các sản phẩm), lập chỉ mục cho trường creatorId. Một sản phẩm lưu trong nhánh /products phải có các nút được chỉ định (.validate). Các sản phẩm cũng chỉ có thể được cập nhật hay xóa bởi người dùng tạo chúng (creatorId == auth.uid). Trên nhánh /userFavorites (lưu danh mục sản phẩm yêu thích theo từng người dùng), người dùng chỉ được hiệu chỉnh danh mục yêu thích của riêng mình ($uid == auth.uid).

Bước 2: Cài đặt chứng thực dùng với Firebase Authentication

- Khai báo thư viện *flutter_dotenv*, *http*, *shared_preferences* và tập tin *.env* trong *pubspec.yaml*:

```
dependencies:
  #...
  flutter_dotenv: ^5.1.0
  http: ^1.1.0
  shared_preferences: ^2.2.2
#...
flutter:
  uses-material-design: true
  assets:
    - .env
  #...
```

- Vào thư mục **auth** đã cho, sao chép 3 thư mục trong đó vào thư mục lib của dự án. Hiệu chỉnh *lib/ui/screens.dart*:

```
...
export 'auth/auth_screen.dart';
export 'auth/auth_manager.dart';

export 'splash_screen.dart';
```

- Hiệu chỉnh *lib/main.dart* thực hiện các tác vụ sau: (1) đọc tập tin .env, (2) tạo và cung cấp đối tượng **AuthManager** và (3) tiêu thụ/sử dụng đối tượng AuthManager để kiểm tra trạng thái đăng nhập của người dùng và trình bày giao diện tương ứng.

```dart
import 'package:flutter/material.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:provider/provider.dart';

import 'ui/screens.dart';

Future<void> main() async {
  // (1) Load the .env file
  await dotenv.load();
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        // (2) Create and provide AuthManager
        ChangeNotifierProvider(
          create: (context) => AuthManager(),
        ),
        ...
      ],
      // (3) Consume the AuthManager instance
      child: Consumer<AuthManager>(
        builder: (ctx, authManager, child) {
          return MaterialApp(
            ...
            home: authManager.isAuth
                ? const SafeArea(child: ProductsOverviewScreen())
                : FutureBuilder(
                    future: authManager.tryAutoLogin(),
                    builder: (ctx, snapshot) {
                      return snapshot.connectionState == ConnectionState.waiting
                          ? const SafeArea(child: SplashScreen())
                          : const SafeArea(child: AuthScreen());
                    },
                  ),
            ...
          );
        },
      ),
    );
  }
}
```

- Hiệu chỉnh widget **AppDrawer** (*lib/ui/shared/app_drawer.dart*), thêm đường liên kết đăng xuất:

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import '../auth/auth_manager.dart';
import '../orders/orders_screen.dart';
import '../products/user_products_screen.dart';

class AppDrawer extends StatelessWidget {
  const AppDrawer({super.key});

  @override
  Widget build(BuildContext context) {
    return Drawer(
      child: Column(
        children: <Widget>[
          ...
          const Divider(),
          ListTile(
            leading: const Icon(Icons.exit_to_app),
            title: const Text('Logout'),
            onTap: () {
              Navigator.of(context)
                ..pop()
                ..pushReplacementNamed('/');
              context.read<AuthManager>().logout();
            },
          ),
        ],
      ),
    );
  }
}
```

- **Lưu ý**: nếu sử dụng **thiết bị Android thật** để chạy ứng dụng thì cần chỉnh quyền truy xuất Internet bằng cách hiệu chỉnh tập tin **android/app/src/main/AndroidManifest.xml**, thêm dòng khai báo sau:

```xml
<manifest ...>
    <application>
    ...
    </application>

    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Nếu dùng thiết bị giả lập thì không cần.

- Chạy ứng dụng và kiểm tra chức năng đăng ký, đăng nhập, đăng xuất. Sau đó, lưu thay đổi vào git và GitHub:

```
git add -u
git add lib
git commit -m "Cai dat chung thuc voi Firebase Authentication"
git push origin master
```

Bước 3: Lưu danh mục sản phẩm lên dịch vụ CSDL của Firebase

- Hiệu chỉnh lớp **Product** thêm định nghĩa hai phương thức **toJson()** và **fromJson()** giúp chuyển đổi qua lại giữa một đối tượng Product và chuỗi JSON.

```
import 'package:flutter/foundation.dart';

class Product {
  ...
  Map<String, dynamic> toJson() {
    return {
      'title': title,
      'description': description,
      'price': price,
      'imageUrl': imageUrl,
    };
  }

  static Product fromJson(Map<String, dynamic> json) {
    return Product(
      id: json['id'],
      title: json['title'],
      description: json['description'],
      price: json['price'],
      imageUrl: json['imageUrl'],
    );
  }
}
```

- Định nghĩa lớp trừu tượng **FirebaseService** lưu giữ những tham số cần thiết để thực hiện các lời gọi REST API (*lib/services/firebase_service*):

```dart
import 'package:flutter/foundation.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';

import '../models/auth_token.dart';

abstract class FirebaseService {
  String? _token;
  String? _userId;
  late final String? databaseUrl;

  FirebaseService([AuthToken? authToken])
      : _token = authToken?.token,
        _userId = authToken?.userId {
    databaseUrl = dotenv.env['FIREBASE_RTDB_URL'];
  }

  set authToken(AuthToken? authToken) {
    _token = authToken?.token;
    _userId = authToken?.userId;
  }

  @protected
  String? get token => _token;

  @protected
  String? get userId => _userId;
}
```

- Tiếp tục hiệu chỉnh lớp **FirebaseService**, thêm phương thức *httpFetch* để tạo các yêu cầu HTTP GET, POST, PUT, PATCH và DELETE:

```dart
import 'dart:convert';
import 'package:http/http.dart' as http;
import '../models/http_exception.dart';
...

enum HttpMethod { get, post, put, patch, delete }

abstract class FirebaseService {
  ...

  Future<Map<String, dynamic>?> httpFetch(
    String uri, {
    HttpMethod method = HttpMethod.get,
    Map<String, String>? headers,
    Object? body,
  }) async {
    Uri requestUri = Uri.parse(uri);
    http.Response response = switch (method) {
      HttpMethod.get => await http.get(
          requestUri,
          headers: headers,
        ),
      HttpMethod.post => await http.post(
          requestUri,
          headers: headers,
          body: body,
        ),
      HttpMethod.put => await http.put(
          requestUri,
          headers: headers,
          body: body,
        ),
      HttpMethod.patch => await http.patch(
          requestUri,
          headers: headers,
          body: body,
        ),
      HttpMethod.delete => await http.delete(
          requestUri,
          headers: headers,
        )
    };

    final json = jsonDecode(response.body) as Map<String, dynamic>?;
    if (response.statusCode != 200) {
      throw HttpException(json!['error']);
    }
    return json;
  }
}
```

- Định nghĩa lớp **ProductsService** chịu trách nhiệm thực hiện các lời gọi REST API đến Firebase để truy xuất và cập nhật dữ liệu (*lib/services/products_service.dart*):
    - Phương thức **fetchProducts()**:

```
import '../models/product.dart';
import '../models/auth_token.dart';

import 'firebase_service.dart';

class ProductsService extends FirebaseService {
  ProductsService([AuthToken? authToken]) : super(authToken);

  Future<List<Product>> fetchProducts({bool filteredByUser = false}) async {
    final List<Product> products = [];

    try {
      final filters =
          filteredByUser ? 'orderBy="creatorId"&equalTo="$userId"' : '';

      final productsMap = await httpFetch(
        '$databaseUrl/products.json?auth=$token&$filters',
      );

      final userFavoritesMap = await httpFetch(
        '$databaseUrl/userFavorites/$userId.json?auth=$token',
      );

      productsMap?.forEach((productId, product) {
        final isFavorite = (userFavoritesMap == null)
            ? false
            : (userFavoritesMap[productId] ?? false);
        products.add(
          Product.fromJson({
            'id': productId,
            ...product,
          }).copyWith(isFavorite: isFavorite),
        );
      });
      return products;
    } catch (error) {
      log('$error');
      return products;
    }
  }
}
```

    - Phương thức **addProduct()**:

```
class ProductsService extends FirebaseService {
  ...

  Future<Product?> addProduct(Product product) async {
    try {
      final newProduct = await httpFetch(
        '$databaseUrl/products.json?auth=$token',
        method: HttpMethod.post,
        body: product.toJson()
          ..addAll({
            'creatorId': userId,
          }),
      );

      return product.copyWith(
        id: newProduct!['name'],
      );
    } catch (error) {
      print(error);
      return null;
    }
  }
}
```

- Hiệu chỉnh lớp **ProductsManager** sử dụng **ProductsService** để tải danh mục sản phẩm và thêm sản phẩm vào CSDL (*lib/ui/products/products_manager.dart*):

```dart
import 'package:flutter/foundation.dart';

import '../../models/auth_token.dart';
import '../../models/product.dart';
import '../../services/products_service.dart';

class ProductsManager with ChangeNotifier {
  List<Product> _items = [];

  final ProductsService _productsService;

  ProductsManager([AuthToken? authToken])
      : _productsService = ProductsService(authToken);

  set authToken(AuthToken? authToken) {
    _productsService.authToken = authToken;
  }

  Future<void> fetchProducts() async {
    _items = await _productsService.fetchProducts();
    notifyListeners();
  }

  Future<void> fetchUserProducts() async {
    _items = await _productsService.fetchProducts(
      filteredByUser: true,
    );
    notifyListeners();
  }

  Future<void> addProduct(Product product) async {
    final newProduct = await _productsService.addProduct(product);
    if (newProduct ≠ null) {
      _items.add(newProduct);
      notifyListeners();
    }
  }
  ...
}
```

- Hiệu chỉnh widget **ProductsOverviewScreen** thực hiện tải danh mục sản phẩm khi khởi tạo. Trong quá trình tải dữ liệu thì hiển thị thanh tiến trình chờ (*lib/ui/products/products_overview_screen.dart*):

```
class _ProductsOverviewScreenState extends State<ProductsOverviewScreen> {
  final _showOnlyFavorites = ValueNotifier<bool>(false);
  late Future<void> _fetchProducts;

  @override
  void initState() {
    super.initState();
    _fetchProducts = context.read<ProductsManager>().fetchProducts();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('MyShop'),
        actions: <Widget>[
          ProductFilterMenu(
            onFilterSelected: (filter) {
              if (filter == FilterOptions.favorites) {
                _showOnlyFavorites.value = true;
              } else {
                _showOnlyFavorites.value = false;
              }
            },
          ),
          ...
        ],
      ),
      drawer: const AppDrawer(),
      body: FutureBuilder(
        future: _fetchProducts,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.done) {
            return ValueListenableBuilder<bool>(
                valueListenable: _showOnlyFavorites,
                builder: (context, onlyFavorites, child) {
                  return ProductsGrid(onlyFavorites);
                });
          }
          return const Center(
            child: CircularProgressIndicator(),
          );
        },
      ),
    );
  }
}
```

- Hiệu chỉnh widget **UserProductsScreen** thực hiện tải danh mục sản phẩm khi khởi tạo. Trong quá trình tải dữ liệu thì hiển thị thanh tiến trình chờ (*lib/ui/products/user_products_screen.dart*):

```
...
class UserProductsScreen extends StatelessWidget {
  static const routeName = '/user-products';

  const UserProductsScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      ...
      drawer: const AppDrawer(),
      body: FutureBuilder(
        future: context.read<ProductsManager>().fetchUserProducts(),
        builder: (ctx, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(
              child: CircularProgressIndicator(),
            );
          }
          return RefreshIndicator(
            onRefresh: () =>
                context.read<ProductsManager>().fetchUserProducts(),
            child: const UserProductList(),
          );
        },
      ),
    );
  }
}
```

- Hiệu chỉnh *lib/main.dart* cập nhật authToken của **ProductsManager** dựa trên giá trị authToken của **AuthManager**. Thay thế:

```
    ChangeNotifierProvider(
        create: (ctx) => ProductsManager(),
    )
```

bằng:

```
    ChangeNotifierProxyProvider<AuthManager, ProductsManager>(
        create: (ctx) => ProductsManager(),
        update: (ctx, authManager, productsManager) {
            // Khi authManager có báo hiệu thay đổi thì đọc lại authToken
            //      cho productManager
            productsManager!.authToken = authManager.authToken;
            return productsManager;
        },
    )
```

- Chạy kiểm tra thêm các sản phẩm. Sau đó, lưu các thay đổi vào git và GitHub:

```
git add -u
git add lib/services
git commit -m "Luu danh muc san pham vao Realtime Database cua Firebase"
git push origin master
```

Bước 4: Cập nhật, xóa sản phẩm lưu trên CSDL của Firebase

- Hiệu chỉnh **ProductsService** (*lib/services/products_service.dart*), thêm hai phương thức để cập nhật và xóa sản phẩm lưu trên CSDL Realtime của Firebase:

```
class ProductsService extends FirebaseService {
  ...

  Future<bool> updateProduct(Product product) async {
    try {
      await httpFetch(
        '$databaseUrl/products/${product.id}.json?auth=$token',
        method: HttpMethod.patch,
        body: product.toJson(),
      );

      return true;
    } catch (error) {
      print(error);
      return false;
    }
  }

  Future<bool> deleteProduct(String id) async {
    try {
      await httpFetch(
        '$databaseUrl/products/$id.json?auth=$token',
        method: HttpMethod.delete,
      );

      return true;
    } catch (error) {
      print(error);
      return false;
    }
  }
}
```

- Hiệu chỉnh **ProductsManager** (*lib/ui/products/products_manager.dart*) cập nhật phương thức *updateProduct* và *deleteProduct*:

```
...
class ProductsManager with ChangeNotifier {
  ...
  Future<void> updateProduct(Product product) async {
    final index = _items.indexWhere((item) => item.id == product.id);
    if (index >= 0) {
      if (await _productsService.updateProduct(product)) {
        _items[index] = product;
        notifyListeners();
      }
    }
  }

  Future<void> deleteProduct(String id) async {
    final index = _items.indexWhere((item) => item.id == id);
    Product? existingProduct = _items[index];
    _items.removeAt(index);
    notifyListeners();

    if (!await _productsService.deleteProduct(id)) {
      _items.insert(index, existingProduct);
      notifyListeners();
    }
  }
}
```

- Hiệu chỉnh **_saveForm** trong lớp **_EditProductScreenState** (*lib/ui/products/edit_product_screen.dart*):

```
Future<void> _saveForm() async {
    ...
    try {
      final productsManager = context.read<ProductsManager>();
      if (_editedProduct.id != null) {
        await productsManager.updateProduct(_editedProduct);
      } else {
        await productsManager.addProduct(_editedProduct);
      }
    } catch (error) {
      if (mounted) {
        await showErrorDialog(context, 'Something went wrong.');
      }
    }
    ...
  }
```

- Chạy kiểm tra cập nhật và xóa sản phẩm. Sau đó, lưu các thay đổi vào git và GitHub:

```
    git add -u
    git commit -m "Cap nhat, xoa san pham tren Realtime Database cua Firebase"
    git push origin master
```

Bước 5: Lưu trữ sản phẩm yêu thích

- Hiệu chỉnh **ProductsService** (*lib/services/products_service.dart*) thêm phương thức lưu trữ sản phẩm yêu thích:

```
class ProductsService extends FirebaseService {
  ...

  Future<bool> saveFavoriteStatus(Product product) async {
    try {
      await httpFetch(
        '$databaseUrl/userFavorites/$userId/${product.id}.json?auth=$token',
        method: HttpMethod.put,
        body: product.isFavorite,
      );

      return true;
    } catch (error) {
      print(error);
      return false;
    }
  }
}
```

- Hiệu chỉnh **ProductsManager** (*lib/ui/products/products_manager.dart*) thêm phương thức thay đổi trạng thái yêu thích của sản phẩm:

```
...
class ProductsManager with ChangeNotifier {
  ...
  Future<void> toggleFavoriteStatus(Product product) async {
    final savedStatus = product.isFavorite;
    product.isFavorite = !savedStatus;

    if (!await _productsService.saveFavoriteStatus(product)) {
      product.isFavorite = savedStatus;
    }
  }
}
```

- Hiệu chỉnh widget **ProductListTile** (*lib/ui/products/product_grid_tile.dart*), thay thế:

```
    onFavoritePressed: () {
        product.isFavorite = !product.isFavorite;
    }
```

bằng:

```
    onFavoritePressed: () {
        ctx.read<ProductsManager>().toggleFavoriteStatus(product);
    }
```

(cần import 'products_manager.dart').

- Kiểm tra chức năng lưu trữ sản phẩm yêu thích. Sau đó lưu thay đổi vào git và GitHub:

```
git add -u
git commit -m "Luu san pham yeu thich len Realtime Database cua Firebase"
git push origin master
```

- Cấu trúc thư mục mã nguồn hiện tại: